מבנים (struct) :

وظيفة هاد الكود انو بيجمع معلومات من المستخدم بعدها بطبعها.

```cpp
const int MAX_GRADES = 5,
          ID_LEN = 10,
          NUM_OF__PERSONS = 4;

struct Person {
    char _id[ID_LEN];
    bool _sex;
    int _shoe_num,
        _grades[MAX_GRADES];
};
```

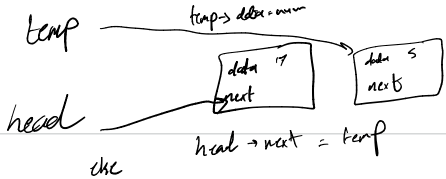
P + _
مش ضروري نحط _ بداية
الكلمة هي تابعة لل struct
بس أفضل للتمييز

; → لازم نحطها مهمة

```cpp
//-----------------------------------------------------
void read_a_person(struct Person &p);
void read_persons(struct Person persons[]);
void print_a_person(const struct Person &p);
void print_persons(const struct Person persons[]);
//-----------------------------------------------------

int main() {
    struct Person persons[NUM_OF__PERSONS];

    read_persons(persons);
    print_persons(persons);
    // ...

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------------
```

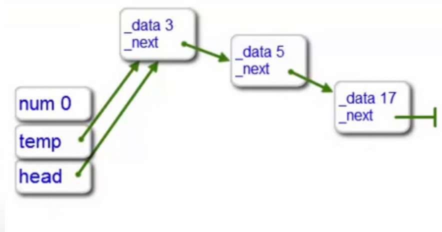
מערך של מבנים,
שמאפשר שמירת מידע
אודות אנשים רבים

```cpp
    // ...

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------------
void read_persons(struct Person persons[]) {
    for (int i = 0; i < NUM_OF__PERSONS; i++)
        read_a_person(persons[i]);
}
//-----------------------------------------------------
void print_persons(const struct Person persons[]) {
    for (int i = 0; i < NUM_OF__PERSONS; i++)
        print_a_person(persons[i]);
}
//-----------------------------------------------------
void read_a_person(struct Person &p) {
    cin >> setw(ID_LEN) >> p._id;
    cin >> p._sex;
    cin >> p._shoe_num;
    for (int i = 0; i < MAX_GRADES; i++)
        cin >> p._grades[i];
}
//-----------------------------------------------------
void print_a_person(const struct Person &p) {
    cout << p._id << " ";
    cout << ((p._sex) ? "FEMALE " : "MALE ");
    cout << p._shoe_num << endl;
    for (int i = 0; i < MAX_GRADES; i++)
        cout << p._grades[i] << " ";
```



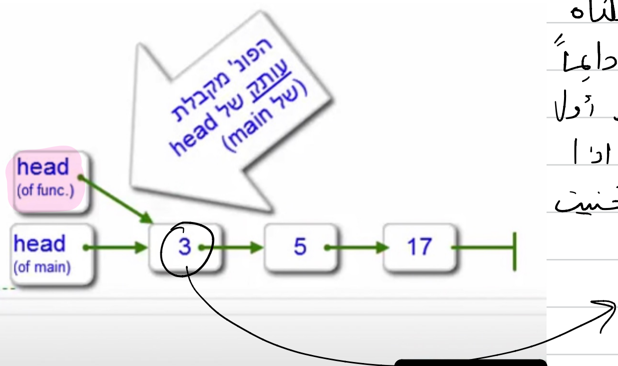
רשימות מקושרות:

```
//-------------------------------------------------
struct Node {
    int _data;
    struct Node *_next;
};
//-------------------------------------------------
void print_list(const struct Node *head);
//-------------------------------------------------
int main() {
    struct Node *head = NULL,
                *temp ;
    int num;

    cin >> num;
    while (num != 0) {
        temp = new (std::nothrow) struct Node;
        if (temp == NULL) {
            cerr << "Cannot allocate memory\n";
            exit(EXIT_FAILURE);
        }

        //(*temp)._data = num;
        temp->_data = num;
        temp->_next = head;
        head = temp;

        cin >> num;
    }

    print_list(head);
```



مرابطة بسيطة
لشيء نعبي ال
ديتا.
اللي هو هان اسمها
head.

```
        }

        //(*temp)._data = num;
        temp->_data = num;
        temp->_next = head;
        head = temp;

        cin >> num;
    }

    print_list(head);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-------------------------------------------------
void print_list(const  struct Node *head) {
    while (head != NULL) {
        cout << head->_data << " ";
        head = head->_next;
    }
    cout << endl;
}
```



[redacted]

الرقم اللي دخلناه
آخر إيش دايماً
راح نطبعو أول
إيشي ، إلا اذا
غيرنا بالتوخنيت
نفسها.

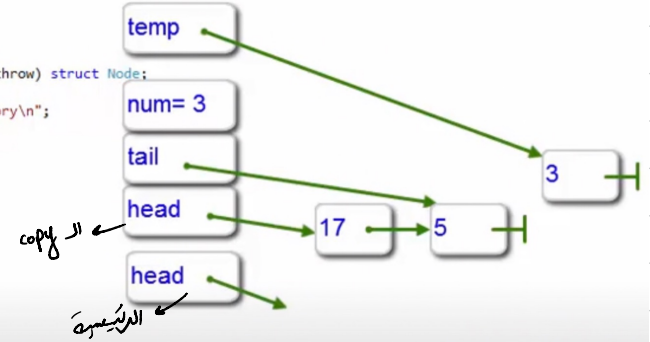
طريقة ثانية بحبي الواجب

```
struct Node {
    int _data;
    struct Node *_next;
};
//-----------------------------------------
struct Node *build_list();
void print_list(const struct Node *head);
//-----------------------------------------
int main() {
    struct Node *head;

    head = build_list();
    print_list(head);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------
```

struct Node הכולל חבר: _data
שמייצג את הנתונים הנשמרים
ברשימה, וחבר: _next המאפשר
את השרשור

num= 15

```
struct Node *build_list() {
    struct Node *head = NULL,
                *tail = NULL;
    int num;

    cin >> num;
    while (num != 0) {
        struct Node *temp = new (std::nothrow) struct Node;
        if (temp == NULL) {
            cerr << "Cannot allocate memory\n";
            exit(EXIT_FAILURE);
        }
        temp->_data = num;
        temp->_next = NULL;
        if (head == NULL)
            head = tail = temp;
        else {
            tail->_next = temp;
            tail = temp;
        }
        cin >> num;
    }
    return head;
}
//-----------------------------------------
void print_list(const  struct Node * head) {
    while (head != NULL) {
        cout << head->_data << " ";
        head = head->_next;
    }
```



רשימה מקושרת ממוינת

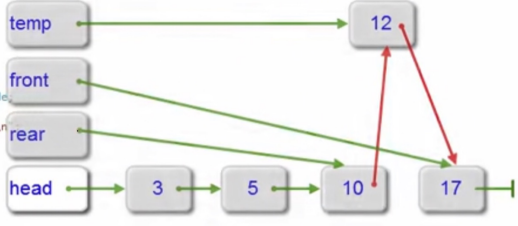
```
//-------------------------------------------------
int main() {
    struct Node *head;

    head = build_sorted_list();
    print_list(head);
    free_list(head);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-------------------------------------------------
```

temp → 12

12

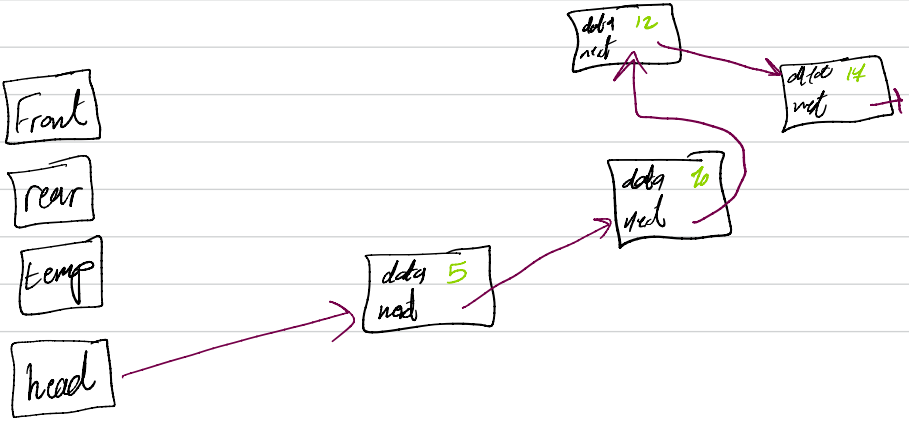
head → 3 → 10 → 17

```
struct Node *build_sorted_list() {
    struct Node *head = NULL,
                *temp,
                *rear, *front ;
    int num;

    cin >> num;
    while (num != 0) {
        temp = new (std::nothrow) struct Node;
        if (temp == NULL) {
            cerr << "Cannot allocate memory\n";
            exit(EXIT_FAILURE);
        }
        temp->_data = num;

        if (head == NULL) {
            head = temp;
            temp->_next = NULL;
        }
        else if (num <= head-> _data) {
            temp -> _next = head ;
            head = temp;
        }
        else {
            rear = head;
            front = rear->_next;
            while (front != NULL && front->_data <= num) {
                rear = front;
                front = front->_next;
            }
            rear->_next = temp;
            temp->_next = front;
        }
        cin >> num;
    }
    return head;
}
```



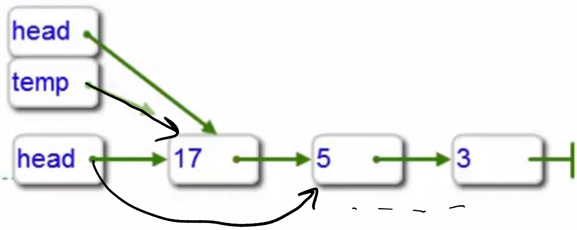
17 10 5 12 0

עתה נשלב את הקופסה עליה מצביע temp במקום המתאים ברשימה שלנו: rear -> _next יצביע על הקופסה החדשה, temp -> _next יצביע כמו front



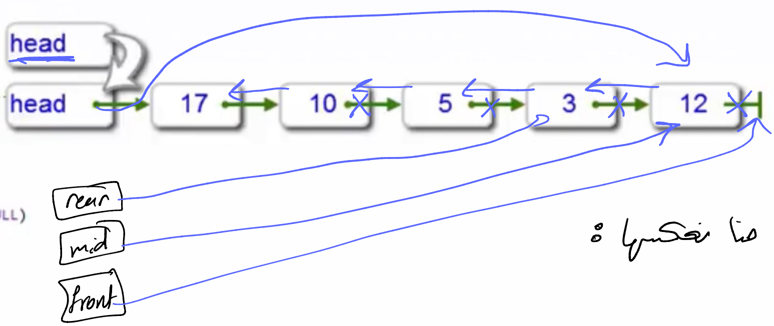
מחיקת רשימה מקושרת

```cpp
//-----------------------------------------------------
void free_list(struct Node *head) {
    struct Node *temp;

    while (head != NULL) {
        temp = head;
        head = head->_next;
        delete temp;
    }
}
//-----------------------------------------------------
void print_list(const  struct Node * head) {
    while (head != NULL) {
        cout << head->_data << " ";
        head = head->_next;
    }
    cout << endl;
}
```



היפוך רשימה מקושרת:

```cpp
int main() {
    struct Node *head;

    head = build_list();
    reverse_list(head);
    print_list(head);
    free_list(head);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------------
void reverse_list(struct Node * &head) {
    struct Node *rear,
                *mid,
                *front;

    if (head == NULL || head->_next == NULL)
        return;

    rear = head;
    mid = rear->_next;
    front = mid->_next;
    while (front != NULL) {
        mid->_next = rear;
        rear = mid;
        mid = front;
        front = front->_next;
    }

    mid->_next = rear;
    head->_next = NULL;
    head = mid;
}
```



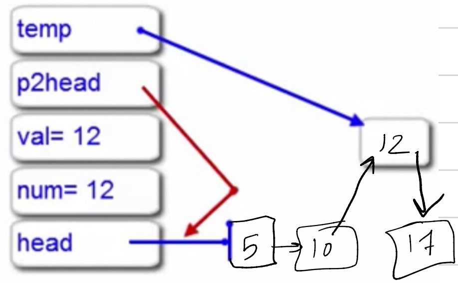
هنا نعكسها :

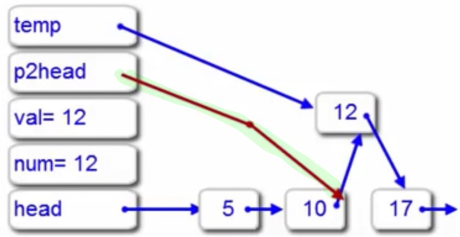
סוג מצביע ל- מצביע:

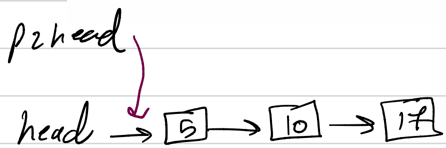
نفس فكرة פוינטר למצביע بس الطريقة بتختلف.

```cpp
}
//------------------------------------------------
struct Node *build_sorted_list2() {
    struct Node *head = NULL;
    int num;

    cin >> num;
    while (num != 0) {
        insert_into_sorted_list(num, &head);
        cin >> num;
    }
    return head;
}
//------------------------------------------------
void insert_into_sorted_list(int val, struct Node **p2head) {
    struct Node *temp = new (std::nothrow) struct Node;

    if (temp == NULL) {
        cerr << "Cannot allocate memory\n";
        exit(EXIT_FAILURE);
    }
    temp->_data = val;

    while (*p2head != NULL && (*p2head)->_data <= val)
        p2head = &((*p2head)->_next);
    temp->_next = *p2head;
    *p2head = temp;
}
```

الקופסה اللي مأشر عليها ال data بعدها.



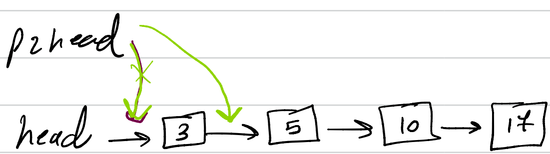




&((*P2head)->_next);

مش ع على وين هي مأشرة أول، بعد 3

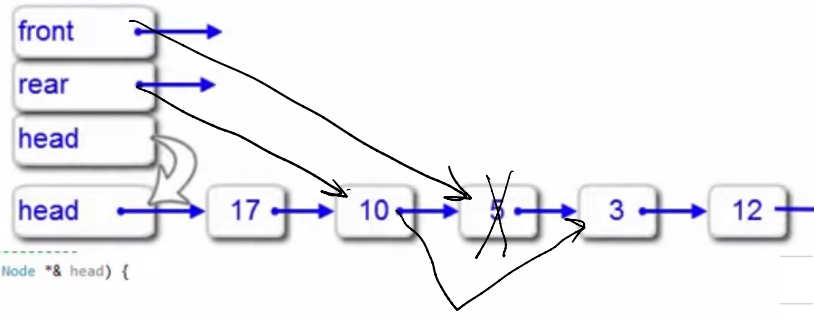
مأشر عند ال next تبعت ال 3



מחיקת איבר מרשימה מקושרת:

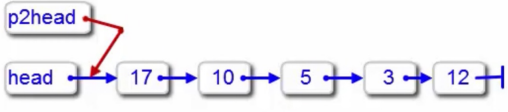
wanted = 5.

```
int main() {
    struct Node *head;
    int wanted;

    head = build_list();
    cin >> wanted;
    delete_from_list(wanted, head);
    print_list(head);
    free_list(head);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
void delete_from_list(int wanted, struct Node *& head) {
    if (head == NULL)
        return;
    if (head->_data == wanted) {
        struct Node *temp = head;
        head = head->_next;
        delete temp;
        return;
    }
    struct Node *rear = head,
                *front = head->_next;
    while (front != NULL) {
        if (front->_data == wanted)
            break;
        rear = front;
        front = front->_next;
    }
    if (front != NULL)
        rear->_next = front->_next;
    delete front;
}
```

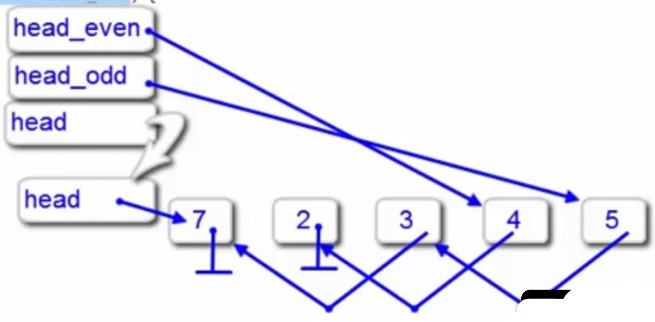


מחיקת איבר מרשימה מקושרת:

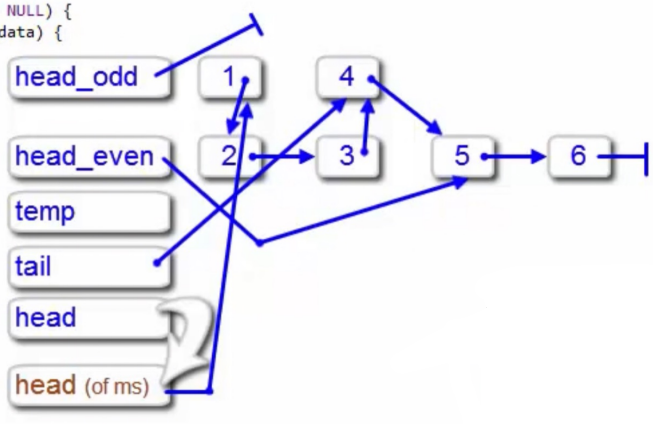
```
int main() {
    struct Node *head;
    int wanted;

    head = build_list();
    cin >> wanted;
    delete_from_list(wanted, &head);
    print_list(head);
    free_list(head);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
void delete_from_list(int wanted, struct Node ** p2head) {
    struct Node *temp;

    while (*p2head != NULL && (*p2head)->_data != wanted)
        p2head = &((*p2head)->_next);
    if (*p2head != NULL) {
        temp = *p2head;
        *p2head = (*p2head)->_next;
        delete temp;
    }
}
```



```
//-----------------------------------------------
void ms(struct Node *& head) {
    struct Node *head_odd, *head_even;

    if (head == NULL || head->_next == NULL)
        return;

    split(head, head_odd, head_even);
    ms(head_odd);
    ms(head_even);
    merge(head, head_odd, head_even);
}
//-----------------------------------------------
void split(struct Node * head,
           struct Node * &head_odd, struct Node * &head_even) {
    struct Node *temp;
    bool odd = true;

    head_odd = head_even = NULL;
    while (head != NULL) {
        temp = head;
        head = head->_next;
        if (odd) {
            temp->_next = head_odd;
            head_odd = temp;
        }
        else {
            temp->_next = head_even;
            head_even = temp;
        }
        odd = !odd;
    }
}
//-----------------------------------------------
void merge(struct Node * &head,
           struct Node * head_odd, struct Node * head_even) {
    struct Node *tail = NULL,
```




```
void merge(struct Node * &head,
           struct Node * head_odd, struct Node * head_even) {
    struct Node *tail = NULL,
                *temp ;

    head = NULL;
    while (head_odd != NULL && head_even != NULL) {
        if (head_odd->_data <= head_even->_data) {
            temp = head_odd;
            head_odd = head_odd->_next;
        }
        else {
            temp = head_even;
            head_even = head_even->_next;
        }
        if (head == NULL)
            head = tail = temp;
        else {
            tail->_next = temp;
            tail = tail->_next;
        }
    }
    if (head_odd == NULL)
        tail->_next = head_even;
    else
        tail->_next = head_odd;
}
```



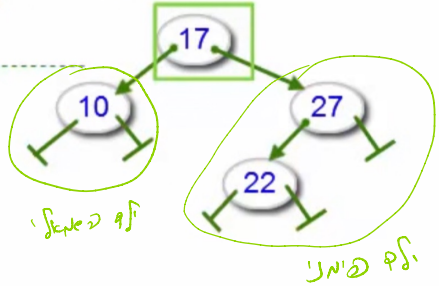
מימוש עצים

מגישת עלי ס

```
struct Node {
    int _data;
    struct Node *_left,
                *_right ;
};
//-------------------------------------------------
void build_bst1(struct Node *&root);
void insert_into_bst1(struct Node *new_node, struct Node *&root);
void print_tree(const struct Node *root);
void free_tree(struct Node *root);

//-------------------------------------------------
int main() {
    struct Node *root = NULL;

    build_bst1(root);
    print_tree(root);
    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-------------------------------------------------
void build_bst1(struct Node *&root) {
    int num;

    cin >> num;
```

א. עץ בינארי עשוי להיות ריק

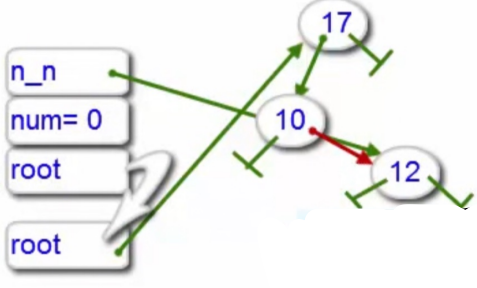
ב. העץ יכיל צומת הקרוי: שורש העץ, ילד שמאלי שהוא עץ בינארי, ילד ימני שהוא עץ בינארי



17 10 12 0

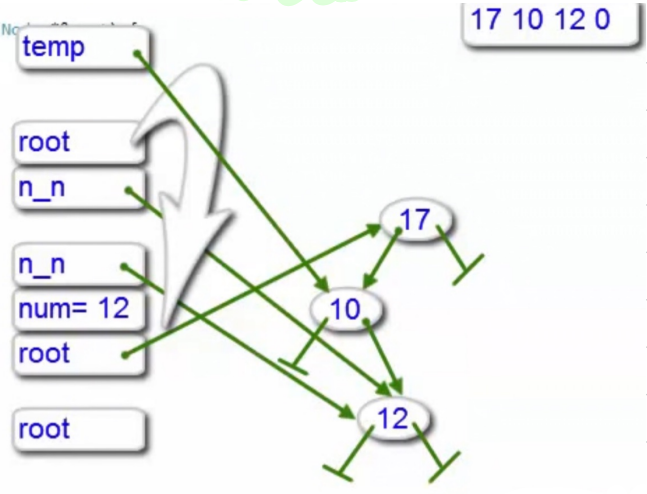
עם ניקוסיה ס

```
//-------------------------------------------------
void build_bst1(struct Node *&root) {
    int num;

    cin >> num;
    while (num != 0) {
        struct Node *new_node = new (std::nothrow) struct Node;
        if (new_node == NULL) {
            cerr << "Cannot allocate memory\n";
            exit(EXIT_FAILURE);
        }
        new_node->_data = num;
        new_node->_left = new_node->_right = NULL;
        insert_into_bst1(new_node, root);

        cin >> num;
    }
}
//-------------------------------------------------
void insert_into_bst1(struct Node *new_node, struct Node *&root)
    if (root == NULL)
        root = new_node;
    else if (root->_data >= new_node->_data)
        insert_into_bst1(new_node, root->_left);
    else
        insert_into_bst1(new_node, root->_right);
}
```



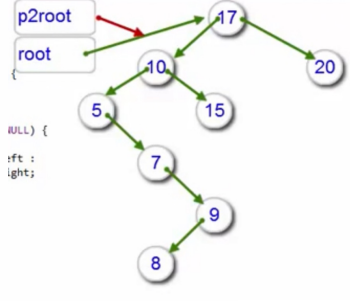
בלי רקורסיה:

```
//-------------------------------------------------
void insert_into_bst2(struct Node *new_node, struct No
    struct Node *temp = root;

    if (root == NULL) {
        root = new_node;
        return;
    }
    while (true) {
        if (new_node->_data <= temp->_data)
            if (temp->_left != NULL)
                temp = temp->_left;
            else
                break;
        else
            if (temp->_right != NULL)
                temp = temp->_right;
            else
                break;
    }
    if (new_node->_data <= temp->_data)
        temp->_left = new_node;
    else
        temp->_right = new_node;

}
//-------------------------------------------------
void free_tree(struct Node * root) {
    if (root != NULL) {
```



כשיש מצביע למצביע:

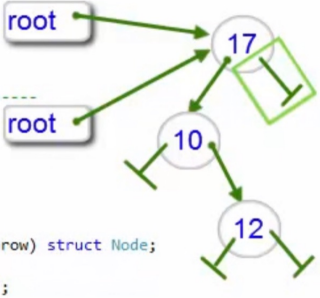
```
struct Node *build_bst3() {
    struct Node *root = NULL;
    int num;

    cin >> num;
    while (num != 0) {
        struct Node *new_node = new (std::nothrow) struct Node;
        if (new_node == NULL) {
            cerr << "Cannot allocate memory\n";
            exit(EXIT_FAILURE);
        }
        new_node->_data = num;
        new_node->_left = new_node->_right = NULL;
        insert_into_bst3(new_node, &root);

        cin >> num;
    }
    return root;
}
//-------------------------------------------------
void insert_into_bst3(struct Node *new_node, struct Node **p2root) {
    while (*p2root != NULL) {
        if (new_node->_data <= (*p2root)->_data)
            p2root = &((*p2root)->_left);
        else
            p2root = &((*p2root)->_right);
    }
    *p2root = new_node;
}
//-------------------------------------------------
```



Print tree

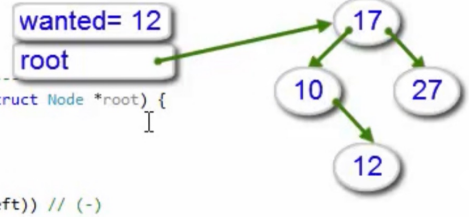
```
//-------------------------------------------------
int main() {
    struct Node *root;

    root = build_bst3();
    print_tree(root);
    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-------------------------------------------------
void print_tree(const  struct Node * root) {
    if (root != NULL) {
        print_tree(root->_left);
        cout << root->_data << " "; // (-)
        print_tree(root->_right);
    }
} //(+)
//-------------------------------------------------
struct Node *build_bst3() {
    struct Node *root = NULL;
    int num;

    cin >> num;
    while (num != 0) {
        struct Node *new_node = new (std::nothrow) struct Node;
        if (new_node == NULL) {
            cerr << "Cannot allocate memory\n";
            exit(EXIT_FAILURE);
```

output: 10 12 17



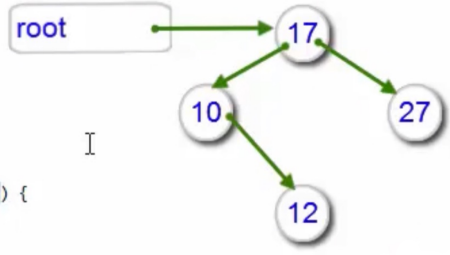
חיפוש עם רקורסיה

```
//-------------------------------------------------
int main() {
    struct Node *root;
    int wanted;

    root = build_tree();

    cin >> wanted;
    if (search_in_tree(wanted, root))
        cout << "+\n";
    else
        cout << "-\n";

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-------------------------------------------------
bool search_in_tree(int wanted, const struct Node *root) {
    if (root == NULL)
        return false;
    if (root->_data == wanted)
        return true;
    if (search_in_tree(wanted, root->_left)) // (-)
        return true;
    return search_in_tree(wanted, root->_right); // (+)
}
//-------------------------------------------------
```



```
//-----------------------------------------------
int main() {
    struct Node *root;
    int wanted;

    root = build_bst();

    cin >> wanted;
    if (search_in_bst(wanted, root))
        cout << "+\n";
    else
        cout << "-\n";

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
bool search_in_bst(int wanted, const struct Node *root) {
    while (root != NULL) {
        if (root->_data == wanted)
            return true;
        else if (wanted < root->_data)
            root = root->_left;
        else
            root = root->_right;
    }
    return false;
}
```

wanted = 12



main

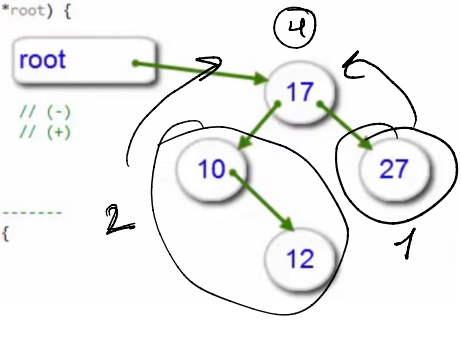
```
//-----------------------------------------------
int main() {
    struct Node *root;
    int wanted;

    root = build_bst();

    cout << count_nodes(root) ;

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
unsigned int count_nodes(const struct Node *root) {
    unsigned int counterl, counterr;

    if (root == NULL)
        return 0;
    counterl = count_nodes(root->_left);     // (-)
    counterr = count_nodes(root->_right);    // (+)

    return counterl + counterr + 1;
}
//-----------------------------------------------
void print_tree(const  struct Node * root) {
    if (root != NULL) {
        print_tree(root->_left);
        cout << root->_data << " "; // (-)
```

→ left (10)
→ left return 0
→ right (12)
→ left return 0
→ right return 0

left(10)
left return 0
right (2)
return 0+0+1

left(10)
return 0+1+1
return 2

ret= in main



2 + 1 + 1 = 4

מציאת עומק עץ

האם הפונ' צריכה להיות רקורסיבית?

יש הכרח ברקורסיה, שכן עבור כל צומת יש למצוא את עומקו של תת-העץ השמאלי **וגם** את עומקו של תת-העץ הימני

```cpp
int main() {
    struct Node *root;
    int wanted;

    root = build_bst();

    cout << find_depth(root);

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
int find_depth(const struct Node *root) {
     int depthl, depthr;

     if (root == NULL)
         return -1;
    depthl = find_depth(root->_left);      // (-)
    depthr = find_depth(root->_right);     // (+)

    return max(depthl, depthr) + 1;
}
//-----------------------------------------------
void print_tree(const  struct Node * root) {
    if (root != NULL) {
        print_tree(root->_left);
        cout << root->_data << " "; // (-)
```

→ main

→ left (10) → return 1

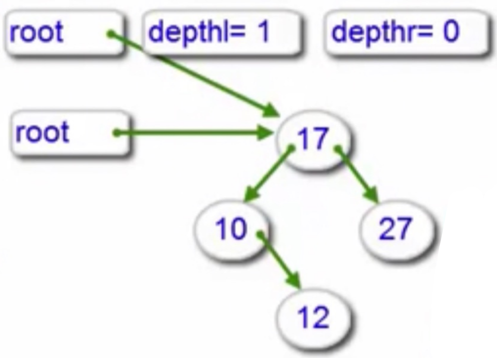
left return (-1)

→ right (12) ← return 0

left return (-1)

right return (-1)

main

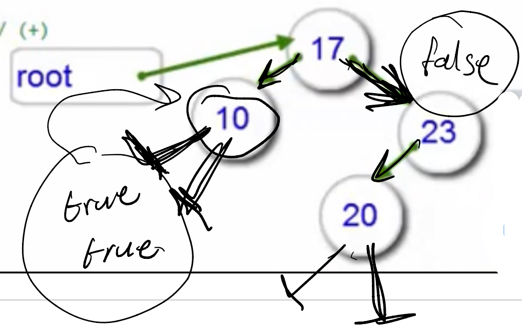


האם לכל צומת פנים יש 2 ילדים?

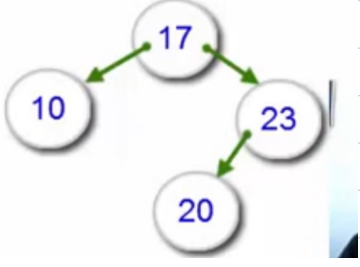
```cpp
int main() {
    struct Node *root;
    int wanted;

    root = build_bst();

    cout << (inner_node_two_children(root) ? "+\n" : "-\n");

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
bool inner_node_two_children(const struct Node *root) {
    bool ok;

    if (root == NULL)
        return true;
    if ((root->_left == NULL && root->_right != NULL) ||
        (root->_left != NULL && root->_right == NULL))
        return false;
    ok = inner_node_two_children(root->_left);
    if (!ok)
        return false;
    ok = inner_node_two_children(root->_right);
    return ok;

    /* return inner_node_two_children(root->_left) &&
              inner_node_two_children(root->_right) ;   */

}
```

האם סכום הערכים משמאל גדול או שווה מימין

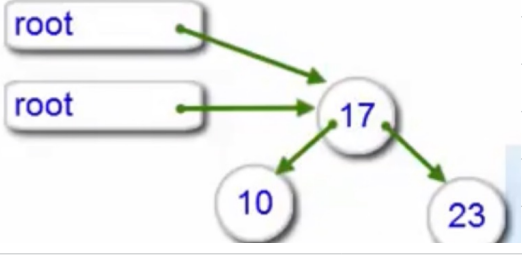
```
int main() {
    struct Node *root;

    root = build_bst();

    cout << (sum_left_smaller_sum_right1(root) ? "+\n" : "-\n");

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-----------------------------------------------
bool sum_left_smaller_sum_right1(const struct Node *root) {
    bool ok;

    if (root == NULL)
        return true;
    if (sum_tree(root->_left) > sum_tree(root->_right))
        return false;
    ok = sum_left_smaller_sum_right1(root->_left);   // (-)
    if (!ok)
        return false;
    ok = sum_left_smaller_sum_right1(root->_right);  // (+)
    return ok;
}
//-----------------------------------------------
int sum_tree(const  struct Node * root) {
    if (root == NULL)
        return 0;
    return sum_tree(root->_left) + root->_data +
           sum_tree(root->_right);
}
```



```
int main() {
    struct Node *root;
    int foo;

    root = build_bst();

    cout << (sum_left_smaller_sum_right2(root, foo) ? "+\n" : "-\n");

    free_tree(root);

    std::cin.get(); std::cin.get();
    return EXIT_SUCCESS;
}
//-------------------------------------------------
bool sum_left_smaller_sum_right2(const struct Node *root, int &sum) {
    bool ok;
    int suml, sumr;

    if (root == NULL) {
        sum = 0;
        return true;
    }
    ok = sum_left_smaller_sum_right2(root->_left, suml);   // (-)
    if (!ok)
        return false;
    ok = sum_left_smaller_sum_right2(root->_right, sumr);  // (+)
    if (!ok)
        return false;
    sum = suml + sumr + root->_data;
    return (suml <= sumr);
}
//
```

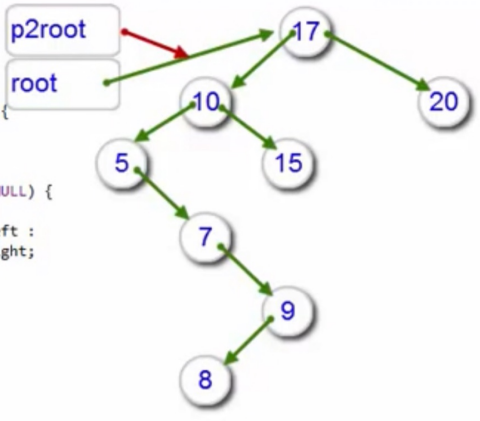



```
//-------------------------------------------------
void free_tree(struct Node * root) {
    if (root != NULL) {
        free_tree(root->_left);      // (-)
        free_tree(root->_right);     // (+)
        delete root;
    }
}
//-------------------------------------------------
```



מחיקת ערך מעץ?

```
void  delete_value(struct Node **p2root, int wanted) {
    struct Node **p2wanted = locate(p2root, wanted);

    if (p2wanted == NULL)
        return;
    if ((*p2wanted)->_left == NULL && (*p2wanted)->_right == NULL) {
        delete (*p2wanted);
        *p2wanted = NULL;
    }
    else if ((*p2wanted)->_left == NULL || (*p2wanted)->_right == NULL) {
        struct Node *temp = *p2wanted;
        *p2wanted = ((*p2wanted)->_left != NULL) ? (*p2wanted)->_left :
                                                   (*p2wanted)->_right;
        delete temp;
    }
    else {
        struct Node **p2prev = find_prev(p2root);
        (*p2wanted)->_data = (*p2prev)->_data;
        struct Node *temp = *p2prev;
        *p2prev = (*p2prev)->_left;
        delete temp;
    }
}
//-------------------------------------------------
struct Node **locate(struct Node **p2root, int wanted) {
    while (*p2root != NULL)
       if ((*p2root)->_data == wanted)
           return p2root;
       else if ((*p2root)->_data > wanted)
           p2root = &((*p2root)->_left);
       else
           p2root = &((*p2root)->_right);

    return NULL;
}
```



צומת אין לו ילדים.

צומת יש לו רק ילד אחד.

צומת עם שני ילדים.

```
struct Node **find_prev(struct Node **p2root) {
    p2root = &((*p2root)->_left);
    while ((*p2root)->_right != NULL)
        p2root = &((*p2root)->_right);
    return p2root;
}
```